Problem 1

BNode.h

```
// COS30008, Problem Set 4, 2021

#pragma once

#include <stdexcept>

template<typename S>
struct BNode
{
    S key;
    BNode<S>* left;
    BNode<S>* right;

    static BNode<S> NIL;

    const S& findMax() const
    {
        if ( empty() )
        {
            throw std::domain_error( "Empty tree encountered." );
        }

        return right->empty() ? key : right->findMax();
    }

    const S& findMin() const
    {
        if ( empty() )
        {
            throw std::domain_error( "Empty tree encountered." );
        }

        return left->empty() ? key : left->findMin();
    }

    bool remove( const S& aKey, BNode<S>* aParent )
    {
        BNode<S>* x = this;
        BNode<S>* y = aParent;

        while ( !x->empty() )
        {
            if ( aKey == x->key )
            {
                break;
            }

            y = x;                                          // new parent

            x = aKey < x->key ? x->left : x->right;
        }

        if ( x->empty() )
        {
```

```
        return false;                                   // delete failed
    }

    if ( !x->left->empty() )
    {
        const S& lKey = x->left->findMax();         // find max to left
        x->key = lKey;
        x->left->remove( lKey, x );
    }
    else
    {
        if ( !x->right->empty() )
        {
            const S& lKey = x->right->findMin();    // find min to right
            x->key = lKey;
            x->right->remove( lKey, x );
        }
        else
        {
            if ( y->left == x )
            {
                y->left = &NIL;
            }
            else
            {
                y->right = &NIL;
            }

            delete x;                                   // free deleted node
        }
    }

    return true;
}

   // PS4 starts here

BNode() : key(S()), left(&NIL), right(&NIL) {
}
BNode(const S& aKey) : key(aKey), left(&NIL), right(&NIL) {
}
BNode(S&& aKey) : key(std::move(aKey)), left(&NIL), right(&NIL) {
}

   ~BNode() {
    remove(NULL, this);
}

bool empty() const {
    return this == &NIL;
}
bool leaf() const {
    return left == &NIL && right == &NIL;
}

size_t height() const {
    if (leaf())
        return 0;
```

```
        return max(left->height(), right->height()) + 1;
    }

    bool insert(const S& aKey) {
        if (aKey == key || empty())
            return false;

        if (aKey < key) {
            if (!left->empty())
                return left->insert(aKey);
            left = new BNode(aKey);
        }
        else {
            if (!right->empty())
                return right->insert(aKey);
            right = new BNode(aKey);
        }
        return true;
    }
};

template<typename S>
BNode<S> BNode<S>::NIL;
```

Output

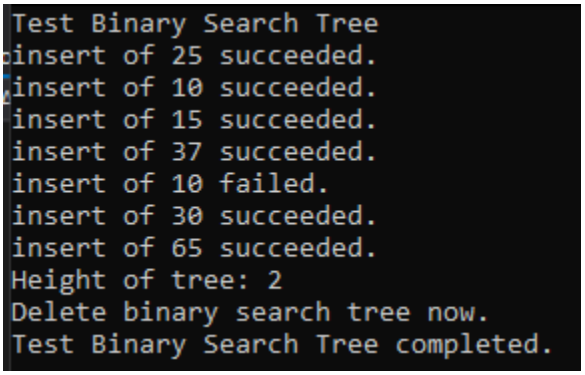
```
Test BNode
insert of 25 as root.
insert of 10 succeeded.
insert of 15 succeeded.
insert of 37 succeeded.
insert of 10 failed.
insert of 30 succeeded.
insert of 65 succeeded.
Height of tree: 2
Delete BNode tree
Test BNode completed.
```

Problem 2

BinarySearchTree.h

```
// COS30008, Problem Set 4, 2021

#pragma once

#include "BNode.h"

template<typename T>
class BinarySearchTreeIterator;

template<typename T>
class BinarySearchTree
{
private:
    BNode<T>* fRoot;

public:

    using Iterator = BinarySearchTreeIterator<T>;

       BinarySearchTree() : fRoot(&BNode<T>::NIL) {
       }
       ~BinarySearchTree() {
              remove(NULL);
       }

       bool empty() const {
              return fRoot->empty();
       }

       bool insert(const T& aKey) {
              if (empty())
                     return fRoot = new BNode<T>(aKey);
              return fRoot->insert(aKey);
       }

       bool remove(const T& aKey) {
              return fRoot->remove(aKey, fRoot);
       }

       size_t height() const {
              return fRoot->height();
       }

       Iterator begin() const {
              return Iterator(const_cast<const BNode<T>*>(fRoot));
       }
       Iterator end() const {
              return Iterator(&BNode<T>::NIL);
       }
};
```

Output


Test Binary Search Tree
insert of 25 succeeded.
insert of 10 succeeded.
insert of 15 succeeded.
insert of 37 succeeded.
insert of 10 failed.
insert of 30 succeeded.
insert of 65 succeeded.
Height of tree: 2
Delete binary search tree now.
Test Binary Search Tree completed.

Problem 3

BinarySearchTreeIterator.h

```
// COS30008, Problem Set 4, 2021

#pragma once

#include <stack>

#include "BNode.h"

template<typename T>
class BinarySearchTreeIterator
{
private:
    const BNode<T>* fBNodeTree;                 // binary search tree
    std::stack<const BNode<T>*> fStack;         // DFS traversal stack

public:

    using Iterator = BinarySearchTreeIterator<T>;

       BinarySearchTreeIterator(const BNode<T>* aBNodeTree)
              : fBNodeTree(aBNodeTree)
       {
              if (!fBNodeTree->empty())
              {
                     const BNode<T>* lNode = fBNodeTree;
                     fStack.push(lNode);

                     while (!lNode->left->empty()) {
                            fStack.push(fStack.top()->left);
                            lNode = lNode->left;
                     }
              }
       }

       const T& operator*() const {
              return fStack.top()->key;
       }

       Iterator& operator++() {
              if (!fStack.empty()) {
                     const BNode<T>* lNode = fStack.top();
                     fStack.pop();
                     if (!lNode->right->empty()) {
                            fStack.push(lNode->right);
                            while (!fStack.top()->left->empty())
                                   fStack.push(fStack.top()->left);
                     }
              }

              if (fStack.empty())
                     fStack.push(&BNode<T>::NIL);

              return *this;
       }
```

```
        Iterator operator++(int) {
                Iterator temp = Iterator(*this);
                ++(*this);
                return temp;
        }

        bool operator==(const Iterator& aOtherIter) const {
                return fStack.top()->key == aOtherIter.fBNodeTree->key;
        }
        bool operator!=(const Iterator& aOtherIter) const {
                return fStack.top()->key != aOtherIter.fBNodeTree->key;
        }

        Iterator begin() const {
                return Iterator(fBNodeTree);
        }
        Iterator end() const {
                return Iterator(&BNode<T>::NIL);
        }
};
```

Output

```
Test Binary Search Tree Iterator DFS
insert of 25 succeeded.
insert of 10 succeeded.
insert of 15 succeeded.
insert of 37 succeeded.
insert of 10 failed.
insert of 30 succeeded.
insert of 65 succeeded.
insert of 8 succeeded.
DFS: 8 10 15 25 30 37 65
Test Binary Search Tree Iterator DFS completed.
```